UMA FERRAMENTA INTERVALAR EM UM AMBIENTE DE PROCESSAMENTO VETORIAL[1]

Rafael Linden Sagula
Alessandra Dahmer
Tiarajú Asmuz Diverio, Dr.
Philippe Olivier Alexandre Navaux, Dr.

Instituto de Informática e CPGCC da UFRGS
Cx.P.15064 CEP 91501-970 - Porto Alegre - RS - Brasil
{dudad,sagula,diverio,navaux}@inf.ufrgs.br
Fax: +55 51 336 5576

ABSTRACT

This paper presents the interval arithmetic in a vector processing environment and its application in order to solve real problems of science and engineering. Thus, we propose a computational tool, *libavi.a*, including interval arithmetic and vector processing. It is part of a project that aims to develop a high-performance arithmetic that provides not only an increase of the processing speed through vectorization, but also accuracy and error control in the operations by using interval arithmetic. In this work it is presented the basic module of this library as well as its interface.

RESUMO

Este artigo apresenta a Aritmética Intervalar num ambiente de processamento vetorial e seu uso na solução de problemas práticos das ciências e engenharias. Para tanto, é proposta uma ferramenta computacional, *libavi.a*, que reúne a aritmética intervalar e o processamento vetorial. Este trabalho é parte de um projeto que visa desenvolver uma aritmética de alto desempenho que proporciona não só aumento da velocidade de processamento via vetorização, mas também exatidão e controle de erros nos cálculos através do uso da aritmética intervalar. Nesse trabalho é apresentado o módulo básico dessa biblioteca e sua *interface*.

[1] Pesquisa desenvolvida com auxílio do CNPq e ProTeM-CC

1 INTRODUÇÃO

A necessidade de ultrapassar as limitações de espaço e tempo levou ao desenvolvimento de máquinas que processam mais velozmente as operações aritméticas e que têm uma capacidade de armazenamento muito grande. Estes supercomputadores têm características de *hardware* que possibilitam o processamento de alto desempenho (vetorial e paralelo). Com isto, tem-se conseguido implementar métodos numéricos que há bem pouco tempo não eram nem sonhados e tornou-se possível a solução de problemas que necessitam de uma grande quantidade de cálculos, conhecidos como problemas de computação de larga escala, que ocorrem nas áreas científicas e militares, nas engenharias, no estudo e exploração de fontes de energia, na medicina, na inteligência artificial e na pesquisa básica ([9]). Entretanto, muitos dos algoritmos numéricos, quando implementados em máquinas digitais, produzem resultados inesperados e, muitas vezes, indesejáveis, pois eles dependem de fatores como: o conjunto sobre o qual os cálculos são efetuados na máquina, a forma como os cálculos são efetuados (aritmética), as formas adotadas para representar números e resultados de operações (tipos de arredondamento), e, a estabilidade do algoritmo e do problema.

Devido ao fato de que a aritmética de ponto-flutuante produz erros na representação de números reais, sentiu-se a necessidade de se ter uma nova ferramenta ou metodologia para o controle de erros. A aritmética intervalar, que utiliza intervalos de números reais para representar valores infinitos, providenciou uma ferramenta para estimar e controlar estes erros automaticamente. Portanto, uma das justificativas de se trabalhar com intervalos é o fato da aritmética real não ser suficientemente confiável em muitos problemas ligados, por exemplo, à física ou química experimental, onde medições não podem ser feitas de forma exata, mas dentro de uma certa "faixa", a qual pode ser representada por intervalos. Mas o sonho de se ter matemática intervalar em supercomputadores ainda permanece! A vantagem de se ter a aritmética intervalar nestas máquinas é o aproveitamento da grande capacidade de processamento que elas possuem.

Foi desenvolvida, pelo grupo de matemática da computação da UFRGS, uma proposta de uma aritmética de alto desempenho, que reuna aritmética intervalar e processamento vetorial ([6, 7, 8]) A desvantagem decorrente da complexidade da forma como as operações intervalares são calculadas é compensada pela segurança e qualidade do resultado e pela "aceleração" decorrente da vetorização das rotinas que implementam as operações intervalares.

Neste artigo, será descrito o módulo básico dessa aritmética. O mesmo é composto por rotinas que operam com intervalos de reais e foi implementado em Fortran 90, no supercomputador Cray Y-MP2E. Esta biblioteca básica torna disponível a aritmética intervalar em uma máquina vetorial e é incorporada em outras bibliotecas ou módulos do ambiente desta ferramenta computacional, que operam com vetores de intervalos, matrizes de intervalos e intervalos complexos.

2 MATEMÁTICA INTERVALAR

O uso da aritmética intervalar foi desenvolvido inicialmente por Moore ([11]). Este ramo da matemática veio se desenvolvendo desde então. A matemática intervalar foi proposta em 1974 por Leslie Fox, combinando diferentes áreas como: aritmética intervalar, análise intervalar, topologia intervalar, álgebra intervalar, entre outras. Ela trata com dados na forma de intervalos numéricos. Surgiu com o objetivo de automatizar a análise do erro computacional e trouxe uma nova

abordagem que permite um controle de erros com limites confiáveis. Na matemática intervalar, em vez de se aproximar um valor real x por um número de máquina, ele é aproximado por um intervalo X, que possui, como limite inferior e superior, números de máquina, de forma que o intervalo contenha x. O tamanho deste intervalo pode ser usado como medida para avaliar a qualidade de aproximação. Os cálculos reais são substituídos por cálculos que usam a aritmética intervalar.

Sejam a e b dois números reais tais que **a** é menor ou igual a **b.** Definimos o intervalo finito **I**, o qual será denotado por **[a, b]** o conjunto ([5]): $I := [a, b] = \{ x \mid \mathbf{a} \leq x \leq \mathbf{b} \}$. A interpretação geométrica de um intervalo, [a,b], pode ser observada no segmento de reta da figura 1.

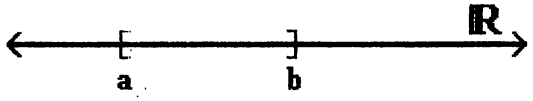

Fig. 1. Representação geométrica de um Intervalo

Seja **I** um intervalo real, chama-se **IR** o conjunto de todos os intervalos reais e é anotado por: IR := { I | I é um intervalo}. Na definição da aritmética intervalar, as operações sobre **IR** devem manter a propriedade do fechamento, ou seja, que o resultado ainda seja um intervalo. Sejam X e Y dois intervalos reais quaisquer, definimos a operação aritmética # pela fórmula:

$$X \# Y := \{x \# y \mid x \in X \text{ e } y \in Y\} \in IR, \text{ onde } \# \in \{ +,-,/,* \}$$

Observa-se que o resultado é um intervalo e que contém o resultado de qualquer operação entre elementos de X com elementos de Y. Observa-se ainda que X/Y somente está definido se o Zero não pertence a Y.

Sejam os intervalos reais X e Y de forma: X:=**[a, b]**, Y:= **[c, d]**. Temos as operações aritméticas definidas como segue:

X + Y := **[a,b] + [c,d] = [a+c, b+d]** (soma);
X - Y := **[a,b] - [c,d] = [a-d, b-c]** (subtração);
X * Y:=**[a,b]*[c,d] = [min{ac,ad,bc,bd},max{ac,ad,bc,bd}** | (produto)
X / Y := **[a,b]/[c,d] = [a,b]*[1/d,1/c]** (se $0 \notin$ **[c,d]**) (divisão)
-X := **[-b,-a]** (pseudo inverso aditivo intervalar)
1/X := **[1/b,1/a]** (pseudo inverso multiplicativo intervalar)

Os intervalos possuem características geométricas importantes: diâmetro, ponto médio, valor absoluto, distância entre intervalos. Além disso, a relação de ordem também é definida sobre intervalos. As principais funções intervalares são: função logarítmica, função exponencial, função quadrada, raiz quadrada, função potência, seno, coseno e tangente. Todas as operações, aspectos geométricos e funções apresentadas neste item foram implementadas no módulo básico da *libavi.a.*

3 PROCESSAMENTO VETORIAL E O CRAY

Um vetor é um conjunto ordenado de elementos. A distinção de vetor e escalar é que escalar é um único elemento, enquanto que vetores são vários elementos. Exemplos de estruturas em Fortran que podem ser representados por vetores são *arrays* de uma dimensão, bem como linhas, colunas e diagonais de *arrays* multidimensionais. Processamento vetorial ocorre quando operações aritméticas e lógicas são aplicadas sobre vetores. A diferença entre o processamento vetorial e o processamento escalar é que o vetorial processa vários elementos no lugar de apenas

um. Em geral, processamento vetorial é mais rápido e mais eficiente do que processamento escalar ([4,13]).

O principal objetivo da vetorização é achar operações seqüenciais que possam ser convertidas em operações vetoriais semanticamente equivalentes, fazendo uso de vantagens do *hardware* vetorial. Em geral, um comando pode ser vetorizado se ele não requer entrada na iteração de uma variável do *loop*, a qual é computada anteriormente no *loop*. O resultado final da vetorização de um *loop* é produzir instruções de máquina que executem grupos de elementos de dados em registradores vetoriais.

O processamento vetorial elimina a maioria dos testes e controle de operações, pois reduz o controle associado à manutenção de variáveis de controle de *loops*. Além disso, ele minimiza o tempo despendido na espera da carga dos operandos e de armazenagem do resultado, pois referencia grupos de locação de memória, usa registradores de múltiplos elementos e reduz o número de instruções em linguagem de máquina que precisam ser carregados, decodificados e executados. Muitas vezes, com o processamento vetorial, são reduzidos problemas de conflito de acesso à memória que ocorrem no processamento escalar.

O Centro Nacional de Supercomputação, localizado na Universidade Federal do Rio Grande do Sul, possui um supercomputador da Cray Research Incorporation, denominado CRAY, modelo Y-MP2E/232, com dois processadores e 256 *Megabytes* de memória principal. A velocidade de pico de cada processador é de 330 *megaflops*, ou seja, 330 milhões de operações de ponto-flutuante por segundo. O sistema operacional é o UNICOS, compatível com o UNIX System V ([3]).

No Cray Y-MP2E as instruções são realizadas por unidades funcionais, as quais implementam um algoritmo ou uma porção do conjunto de instruções. Elas são independentes logicamente e podem operar simultaneamente. No Cray, os processamentos escalar e vetorial podem ser efetuados sobre os dados.

O processamento escalar ocorre seqüencialmente e usa um operando ou um par de operandos para produzir um único resultado. Processamento escalar é efetuado usando registradores escalares (tipo S). Várias unidades funcionais são unicamente dedicadas a processamento escalar. Algumas, como as unidades funcionais de ponto-flutuante, podem ser compartilhadas com operações vetoriais.

O processamento vetorial admite que uma simples operação possa ser realizada concorrentemente sobre um conjunto (ou vetor) de operandos, repetindo a mesma função para produzir uma série de resultados. Processamento vetorial é efetuado em registradores vetoriais (tipo V) e possui unidades funcionais só para este tipo de processamento, além de poder compartilhar as unidades funcionais de ponto-flutuante com o processamento escalar.

Uma dificuldade no processamento vetorial do Cray Y-MP2E identificada é decorrente do fato de que o sistema de representação dos números em ponto-flutuante no Cray não segue o padrão de aritmética binária de ponto-flutuante da IEEE-754. A aritmética de ponto-flutuante do supercomputador Cray não possui, por exemplo, os arredondamentos direcionados. Os tipos de dados disponíveis no Cray são: inteiros, reais, reais em dupla precisão, complexos e complexos em dupla precisão ([4]).

A linguagem utilizada na implementação da biblioteca intervalar vetorial foi o Fortran 90. O mesmo é uma extensão do Fortran padrão([2,10]). Ele contém todo o Fortran 77, ou seja, todos os programas escritos em Fortran 77 são compilados pelo Fortran 90 e, além das principais características existentes no Fortran 77, como sintaxe de vetores, *arrays* automáticos, subprogramas recursivos, linha de inclusão e os comandos estruturados *do-end do* e *do-while*, introduz uma nova forma para programas fonte, números complexos com dupla precisão, atribuição de ponteiros, *arrays* dinâmicos, estruturas, módulos, especificações de interface, várias novas funções intrínsecas e a possibilidade da definição de novos tipos de dados ([2] e [10]).

Novos tipos de dados complexos em dupla precisão são padrões no Fortran 90. Outra inovação importante é a possibilidade de definir-se tipos derivados de dados. Os tipos derivados lembram os registros em Pascal. Eles são definidos por estruturas que iniciam por *type* e terminam por *end type*. O símbolo de porcentagem **%** é usado para referenciar componentes do tipo derivado, como por exemplo intervalos de reais, vetores e matrizes de intervalos, que são definidos e referenciados na figura 2.

```
type interval
        real :: inf, sup
end type interval
type (interval), dimension(n) :: vector_interval
type (interval), dimension (n,m) :: matrix_interval
vector_interval(5) % inf= 3.5
vector_interval(5) % sup = 4.6
vector_interval(6)= interval(3.7 , 4.2)
```

Fig. 2 Definição e referência aos novos tipos em Fortran 90

Uma das limitações do Fortran 77 no uso de funções residia no fato que elas eram limitadas aos tipos pré-definidos de dados. O resultado de uma função era apenas um valor. Não se poderia utilizar funções para intervalos, pois para uma função intervalar teriam de ser produzidos dois valores, um para cada extremo do intervalo resultante.

Em Fortran 90, é possível não só a definição de novos tipos de dados, como também criar funções para estes novos tipos de dados.

As novas funções podem ser renomeadas a símbolos ou operadores usuais da linguagem ou da matemática. Elas podem ser utilizadas em expressões de forma análoga à notação matemática, o que simplifica a programação e o entendimento dos programas.

No Fortran 90, pode-se criar módulos. Eles são entidades únicas, que agrupam tanto a definição de um tipo de dado como as funções e rotinas que o manipula. São definidas ainda no módulo as interfaces, facilitando a utilização dessas funções, pois no programa, apenas é incluída a linha *use module_name*, tornando acessível tudo que está contido no módulo.

Um módulo pode ter partes visíveis fora dele e partes somente internas ao módulo, que não devem ser usadas fora dele. Isso se faz através do comando *private lista*. Assim, tudo que estiver na lista será visível apenas dentro do módulo.

Um módulo pode, ainda, usar outros módulos, criando uma certa hierarquia entre eles, que não poderá ser quebrada. São duas as regras básicas para o uso de módulos: um módulo não pode usar a si mesmo e um módulo não pode usar outro que o utilize.

Uma observação importante é que, se um módulo Y usa outro módulo X e, um módulo Z usa o módulo Y, as funções do X não estão acessíveis ao módulo Z, ou seja, não há transitividade entre os módulos; para usar as funções do módulo X, o módulo Z deve incluir o comando *use X*. A estrutura do comando de criação de módulo é dada pela figura 3.

```
module nome
        use...
        type...
        interface...
        private...
contains
        function...
        subroutine...
end module
```

Fig. 3 Estrutura do comando de criação de módulos

4 A BIBLIOTECA DE ROTINAS INTERVALARES

A biblioteca intervalar vetorial, *libavi.a*, foi implementada em Fortran 90, no supercomputador Cray Y-MP2E. A biblioteca básica implementa rotinas de transferência, operações aritméticas intervalares, operações relacionais e as principais funções intervalares. Esta biblioteca faz parte de um projeto que visa o desenvolvimento de uma aritmética de alto desempenho.

O tipo intervalo foi definido usando-se a possibilidade de definir novos tipos de dados em Fortran 90. Com a definição do tipo intervalo foram definidos operadores para estes tipos. Essa definição é feita na interface de biblioteca. Nesta interface são definidas as rotinas que implementam cada operação. Os arredondamentos para cima e para baixo não estão disponíveis no Cray, por isso foram feitas rotinas que não arredondam, mas sim inflacionam os extremos. As rotinas, *infla_down* e *infla_up*, foram implementadas como mostra a figura 4.

```
function infla_down (A) result (C)
        real:: A,C
        C = NEAREST(A, -1.0)
end function infla_down
```

```
function infla_up (A) result (C)
        real:: A,C
        C = NEAREST(A, +1.0)
end function infla_up
```

Fig. 4 Implementação dos arredondamentos

Foi usada a função *nearest*, que retorna o número de máquina imediatamente anterior ou posterior, dependendo do argumento (- ou +). Com isto pode-se inflacionar o limite inferior para baixo e o superior para cima, garantindo que o intervalo contenha todos as soluções possíveis.

As rotinas implementadas podem ser divididas em cinco grupos:

- rotinas de transferência,
- operações relacionais,
- operações entre conjuntos,
- operações aritméticas,
- funções intervalares.

Todas as rotinas tem o seu nome precedido pela letra s. Esta padronização foi feita para indicar o tipo de dado com que elas operam, de acordo com os padrões adotados pela Cray.

Entre as rotinas de transferência estão as rotinas que calculam particularidades geométricas dos intervalos como diâmetro, módulo, ponto médio. As rotinas deste grupo são *sintpt, sintval, sinf, ssup, smod, sdis, sdia, smed.*

As rotinas de operações relacionais são as rotinas que relacionam intervalos, fazendo comparações entre eles. As rotinas deste grupo são: *sequ, sless, sgre, sleq, sgeq, rins, sneq*. As operações realizadas pelas rotinas de operações entre conjuntos são a união e intersecção (*suni, sinter*).

Todas as operações aritméticas e funções intervalares descritas no item 2 foram implementadas (*sadd, sneg, ssub, smult, sinv, sdiv*). Os extremos são inflacionados após as operações através das rotinas *infla_down* e *infla_up*.

Sobrecarga de funções e operadores é uma das facilidades da biblioteca de rotinas intervalares. A sobrecarga possibilita que o usuário utilize o nome genérico de uma função, independente do tipo de argumentos desta função. A tarefa de identificar a rotina correta que opera com tais argumentos é transferida ao compilador. Isto facilita a programação. Foram sobrecarregados os símbolos das operações aritméticas de soma, subtração, multiplicação, divisão e os símbolos relacionais existentes no Fortran 90. A tabela 1 mostra a relação de rotinas implementadas, nos diferentes módulos, que podem ser chamadas pelo operador +.

Tabela 1 Sobrecarga do operador +

Símbolo	Rotinas	Argumentos	Módulo
+	sadd	s1, s2	Básico
	smt	s1	Básico
	svadd	sv1, sv2	MVI
	svmt	sv1	MVI
	smadd	sm1, sm2	MVI
	smmt	sm1	MVI
	scadd	sc1, sc2	CI
	scmt	sc1	CI
	radds	r1, s1	MVI
	saddr	s1, r1	MVI
	raddsv	r1, sv1	MVI
	svaddr	sv1, r1	MVI
	raddsm	r1, sm1	MVI
	smaddr	sm1, r1	MVI
	saddsv	s1, sv1	MVI
	svadds	sv1, s1	MVI
	saddsm	s1, sm1	MVI
	smadds	sm1, s	MVI
	rvaddsv	rv1, sv1	MVI
	svaddrv	sv1, rv1	MVI
	rmaddsm	rm1, sm1	MVI
	smaddrm	sm1, rm1	MVI

O **uso de arrays dinâmicos na definição de vetores e matrizes de intervalos** se constitui em outra característica da linguagem Fortran, que foi incorporada às características da biblioteca de rotinas intervalares. Ela contribui para a simplificação e facilidade de uso da biblioteca. *Arrays* dinâmicos não apenas permitem um uso otimizado da memória, através do uso de vetores e matrizes de tamanho variado, garantindo alocação e liberação do espaço da memória, como também facilitam a programação, uma vez que o usuário apenas utiliza a sua definição segundo a compatibilidade do sistema. O sistema não impõe restrições desnecessárias e nem exige conhecimento prévio de limites. Quando o usuário declara um vetor ou matriz de intervalos no seu programa aplicativo é que o tamanho ou o número de elementos é definido. Toda a referência a este vetor ou matriz de intervalos leva consigo o tamanho e o tipo dos elementos, inclusive no momento de leitura e impressão de vetores e matrizes intervalares.

Uma característica decorrente do tipo de processamento do supercomputador Cray é a **leitura de dados somente de arquivos**. Adotou-se que intervalos podem ser lidos de arquivos ou atribuídos no programa. Para tanto, foram implementadas as funções de transferência. A impressão de intervalos também deve ser feita em arquivos. Portanto, para resumir, as rotinas *read* e *write* trabalham com arquivos. Para a utilização de arquivos de dados na leitura de tipos intervalares é necessário, no início do programa, definir e abrir o arquivo de dados *nome.dat*.

A extensão do comando *read* para intervalos é feita por: **call sread(5, s1)**. Como *sread* é uma subrotina, deve ser referenciada pelo comando *call*. O formato dos dados no arquivo é livre, para simplificar o uso para os usuários, mas cada linha do arquivo deve conter apenas um intervalo, que corresponde a dois valores reais, onde o primeiro será atribuído a extremo inferior e o segundo ao extremo superior do intervalo.

Para leitura de vetores de intervalos, a extensão é dada por **call svread(5, sv1)**. Na referência da rotina *svread*, a variável sv1 é do tipo *svector*, cuja dimensão foi definida na declaração do programa do usuário. Mais uma vez se convencionou que a cada linha será lido, em formato livre, dois valores reais, correspondendo o primeiro ao extremo inferior e o segundo ao extremo superior. A rotina só será validada se todos os elementos forem intervalos, ou seja, se a leitura dos valores dos intervalos de cada componente do vetor for validada.

Da mesma forma, a rotina *smread* realiza a leitura dos valores intervalares de cada componente da matriz de intervalos. Os valores lidos corresponderão a um intervalo por linha do arquivo, sendo que os valores serão introduzidos por colunas, ou seja, primeiro serão lidos os intervalos que compõem a primeira coluna da matriz, depois os da segunda coluna e, assim por diante, até a última coluna. A linha de comando: **call smread(5, sm1)** é uma referência da rotina.

A rotina que implementa a leitura de intervalos complexos é a *scread* e segue o mesmo princípio, mas para cada intervalo complexo devem ser lidos quatro valores reais, correspondendo o primeiro para o extremo inferior real, o segundo para o extremo superior real, o terceiro para o extremo inferior imaginário e o último, o extremo superior imaginário.

As rotinas de impressão de tipos intervalares são extensões do comando *write* para valores reais, o qual é referenciado por **write(6, *) r1,..,rn**. Esta referência contém o uso mais simples da rotina, sendo '6' a indicação de arquivo de saída padrão e '*' a indicação de uso do formato livre. As variáveis r1 a rn são do tipo real.

A rotina que imprime intervalos reais é a *swrite*. Como é do tipo subrotina, uma referência deve ser precedida pelo comando *call*, ou seja, **call swrite(6, s1)**. Os valores são separados por vírgula e ficam entre colchetes. Sempre é impresso um intervalo por linha.

A impressão de um vetor de intervalos é realizada pela rotina *svwrite* e são impressos os intervalos elementos em formato livre de reais, um intervalo por linha. Ao final do vetor é impressa uma linha em branco para indicar o final do vetor. Uma referência da rotina é da forma **call svwrite(6, sv1).** A impressão de matrizes de intervalos se dá por colunas. A cada bloco de intervalos, correspondentes a colunas, são impressos tantos intervalos quanto o número de linhas da matriz. Ao final de cada coluna é impressa uma linha em branco e, ao final da matriz, são impressas duas linhas em branco. A rotina que imprime matrizes é denominada *smwrite*. Uma referência é dada por **call svwrite(6, sm1).**

A impressão de intervalos complexos se dá pela rotina *scwrite*. Ela imprime dois intervalos entre parêntesis, separados por vírgula. Cada intervalo possui dois reais separados por vírgula e entre colchetes. Portanto, são impressos quatro valores reais para cada intervalo complexo, onde os dois primeiros correspondem aos extremos do intervalo da parte real e os dois últimos ao intervalo da parte complexa. O número imaginário i não é impresso, uma vez que o intervalo complexo é dado na forma de par ordenado de intervalos. Uma referência é dada por **call svwrite(6, sc1)** .

Na documentação ([7,8]) das rotinas da biblioteca intervalar foi usado o formato usado pela Cray Research, Inc. O formato de documentação, padronizado pela Cray, que foi usado para documentar a *libavi.a* é descrito abaixo:

NOME	Contém o nome e as entradas da rotina
SINOPSE	Descreve a sintaxe das entradas. [ ] indicam que o componente é opcional. Elipses indicam que o comando pode ser repetido
DESCRIÇÃO	Descreve a rotina com detalhes
IMPLEMENTAÇÃO	Contém detalhes para o uso da rotina em máquinas ou sistemas operacionais específicos
STANDARDS	Notas relevantes a respeito da rotina que está sendo descrita
NOTAS	Pontos ou itens de particular importância
CUIDADOS	Descreve ações que podem danificar os resultados da rotina
EXEMPLOS	Contém exemplos de uso
ARQUIVOS	Lista os arquivos que cada parta da rotina usa
VALORES RETORNADOS	Descreve possíveis erros retornados
MENSAGENS	Descreve informações, diagnósticos e mensagens e erros que possam aparecer.
VER TAMBÉM	Lista outras rotinas que tenham ligação com a rotina.

Fig. 5 Itens da documentação

O objetivo desta padronização, de acordo com as normas da Cray, é que esta biblioteca possa ser amplamente utilizada e que os usuários tenham a sua disposição uma documentação bem acessível, no formato de outras documentações às quais já estejam acostumados.

5 EXEMPLOS

Um programa simples, que serve de exemplo para uso da interface da biblioteca *libavi.a* é dado na figura 6. As partes em negrito na figura 6 representam o uso da interface. O mesmo programa, escrito sem sobrecarga, é mostrado na figura 7.

```
program exemplo1
use basico
type (interval) :: A,B,C
B  = sintpt(0.2)
C  = sintpt(0.3)
A  = B  + C
call swrite(6,A)
end program exemplo1
```

Fig. 6 Exemplo de uso de interface

```
program exemplo2
use basico
type(interval) :: A,B,C
call scopy(B, sintpt(0.2))
call scopy(C,sintpt(0.3))
call scopy(A, sadd(B,C))
call swrite(6,A)
end program exemplo2
```

Fig. 7 Exemplo sem o uso de interface

O compilador Fortran 90 informa ao usuário, durante a compilação, do como ele está resolvendo cada símbolo de interface. Isso pode ser habilitado através da diretiva de compilação *-m 0*.

Há, também, a possibilidade de total abstração em relação ao tamanho dos vetores intervalares que são usados em qualquer operação envolvendo interface. Isso não significa que os tamanhos possam não ser compatíveis. O exemplo da figura 8 representa essa situação.

```
program exemplo3
use basico
use mvi
type (interval), dimension(3,3) :: A,B,C
B  = svintpt(0.2)
C  = svintpt(0.3)
A  = B  + C
call svwrite(6,A)
end program
```

Fig. 8 Exemplo de uso de arrays unidimensionais intervalares com interface

Pode-se perceber que não foi necessário informar o tamanho dos arrays para a rotina de soma. Dessa forma, foi possível definir uma interface eficiente para essa rotina. Isso se deu através do recurso de arrays dinâmicos, que permitem às rotinas alocarem tanta memória quanto necessitam em tempo de execução.

6 A ARITMÉTICA DE ALTO DESEMPENHO - CONCLUSÕES

A biblioteca de rotinas intervalares básica desenvolvida neste trabalho é apenas um dos módulos da proposta de Aritmética de Alto Desempenho, descrita em [6,13], onde são apresentados detalhes da ferramenta de alto desempenho a ser implementada no supercomputador Cray Y-MP2E.

Para se ter uma aritmética de alta exatidão é necessário que os números sejam representados segundo o padrão binário de aritmética de ponto-flutuante do IEEE-754; que se

tenha o controle de erros através dos arredondamentos direcionados; que as operações aritméticas básicas em ponto-flutuante com arredondamentos sejam definidas segundo a Regra Geral de Kulisch ([9]). Além disto, é necessário se ter a aritmética intervalar, a qual se utiliza dos arredondamentos para sua implementação. Por fim, é necessário que se tenha um produto escalar que produza resultados exatos ou o mais próximo possível, ou seja, que o valor calculado seja diferente do valor exato apenas no último dígito significativo. Este produto escalar é conhecido como produto escalar exato ou ótimo.

A aritmética de alto desempenho foi definida como sendo uma aritmética que produz resultados intervalares contendo a solução exata e que se utiliza das vantagens dos supercomputadores como, por exemplo, processamento vetorial. O uso do processamento vetorial, apesar de polêmico por afetar a ordem dos cálculos gerando uma maior instabilidade, pode ser utilizado de forma controlada visando um ganho computacional, ou seja, alto desempenho.

Deve-se destacar, nessa biblioteca, a limitação da inexistência ou não observância do padrão de aritmética binária da IEEE-754 onde, entre outras coisas, existem definidos quatro tipos de arredondamentos, entre eles, os arredondamentos direcionados para baixo e para cima, necessários na implementação da aritmética intervalar de máquina. Para suprir tal deficiência, utilizou-se o artifício denominado inflação, onde os arredondamentos são conseguidos através da função *nearest*, que calcula o número de máquina mais próximo, tanto para cima quanto para baixo. Isto inflaciona o intervalo que contém o resultado, porém garante que o valor correto se encontre dentro dele. Outra limitação desta proposta reside no fato de que não se tem o produto escalar exato. A implementação de tal operação exige registradores maiores do que os disponíveis no Cray.

O projeto de documentação da *libavi.a* é consistente com o padrão da *Cray Research Inc*, pois se pretende que o *software* seja utilizado por usuários do supercomputador Cray e, com isso, o uso de intervalos seja difundido em diversas áreas que exigem alta exatidão nos cálculos, como nas engenharias, física e química. Além disso, espera-se que com essa biblioteca sejam desenvolvidos novos trabalhos, como por exemplo:

- Desenvolvimento de novos métodos intervalares e implementação, uma vez que se tem a aritmética intervalar e rotinas básicas disponíveis no supercomputador;
- Estudos da viabilidade e validade do desenvolvimento de unidades funcionais que operem com aritmética intervalar em supercomputadores vetoriais;
- Estudo da Implementação do produto escalar ótimo no Cray, a fim de se ter uma aritmética de alta exatidão e algoritmos com verificação automática do resultado;
- Desenvolvimento de pesquisa análoga para o ambiente de computadores paralelos, baseados em *Transputers*, analisando o paralelismo das operações intervalares.

A validação das rotinas implementadas se deu pela realização de vários testes que verificavam a maioria dos casos das operações aritméticas. Uma vez validadas as operações aritméticas intervalares, tem-se automaticamente as demais rotinas validadas, pois estas se utilizam das rotinas das operações aritméticas. A validação dos resultados se deu pelo desenvolvimento de vários programas-teste desenvolvidos, tanto utilizando biblioteca desenvolvida em Fortran 90, como também em Pascal-XSC (linguagem onde a alta exatidão está disponível).

Por fim, a existência dessa ferramenta computacional deverá incentivar e permitir o desenvolvimento de estudos de conversão de métodos reais para intervalares em diversas áreas.

7 BIBLIOGRAFIA

[1] CLAUDIO,D.M; MARINS,J.M. **Cálculo numérico computacional.** São Paulo, Atlas, 1989.

[2] CRAY RESEARCH, Inc. **CF90 Fortran Language Reference Manual.** v.1, n.sr-3902, versão 1.0, 1994.

[3] DAHMER, A et al. **Supercomputador Cray: arquitetura, características e acesso remoto**. Porto Alegre: CPGCC da UFRGS, 1993. RP-210. (março). 86p.

[4] DAHMER,A.; DIVERIO,T.A. Uma análise das propriedades algébricas dos números no Cray. In: **CNMAC**, XVI, Uberlândia, MG, set. 6-9, 1993. p.108

[5] DIVERIO, T.A et al. **Introdução a teoria dos intervalos.** Porto Alegre: UFRGS, 1992. RP-172.

[6] DIVERIO, T. A. **Uso efetivo da matemática intervalar em supercomputadores vetoriais.** Porto Alegre: CPGCC da UFRGS, 1995. 291p. (Tese de doutorado).

[7] DIVERIO, T. A. et al. **LIBAVI.A Biblioteca de Rotinas intervalares - Manual de utilização.** Porto Alegre: CPGCC da UFRGS, 1995. 350p.

[8] DIVERIO, T. A. et al. **LIBAVI.A Biblioteca de Rotinas intervalares - Verificação e validação.** Porto Alegre: CPGCC da UFRGS, 1995. 117p.

[9] KULISCH,U; MIRANKER,W.R **Computer arithmetic theory and pratice** New York: Academic Press, 1987.

[10]METCALF, M; RED,J. - **Fortran 90 explained.** Oxford, Oxford University Press, 1994.

[11] MOORE, RAMON,E - **Interval Analysis**. Englewood Cliffs, Prentice Hall, 1966.

[12] NAVAUX, P.O.A Processadores Pipeline e processamento vetorial. **Escola de Computação**, VII,USP São Paulo, 12-20, Jul, 1990.

[13] DIVERIO, T. A.; NAVAUX, P. O. A.; CLAUDIO, D. M. Alto desempenho e eficiência em processamento numérico. In: **SBAC-PAD**, VII, Canela, jul.29, ago.4, 1995. Porto Alegre: UFRGS/SBC, 1995. pp.257-269.